CSC2001F 2017 Assignment: Graphs

Introduction

This assignment concerns using directed graphs to develop a simulation of an emergency medical ervice; of ambulances, hospitals, and medical emergencies.

- Hospitals have emergency units.
- Ambulances are stationed at hospitals.
- The road network that connects hospitals and victim locations can be modelled with a weighted directed graph.
- Emergency calls are received from victims at particular locations.
- Ambulances must be routed to victims and then on to hospitals.
  (An ambulance can only take a victim to the hospital at which it is stationed.)

Part one of the assignment will be automatically marked, while part two will be manually marked.

Please note that the Automarker will generate random data, so you will not have the same situation every time you run it.

Part one: round-trip simulation

Given an incoming call, the problem is to identify the ambulance that can make the lowest cost round trip to the victim and then back to the hospital at which it is stationed.

You will write a program called '`SimulatorOne.java`' that accepts as input a file containing data on roads, hospitals, and incoming call events. The program will then, for each call, calculate and output details of the (best) shortest round trip.

Your program will use Dijkstra's algorithm to calculate the lowest cost path between victims and hospitals.

A small consideration: since the graph is directed, the lowest cost route to a victim may not be the lowest cost route back to the hospital (in fact, that route may not exist).

Input format:

<number of nodes><newline>
{<source node number> {<destination node number> <weight>}<newline>}**
<number of hospitals><newline>
{<hospital node number>}<newline>*
<number of victims><newline>
{<victim node number>}<newline>*

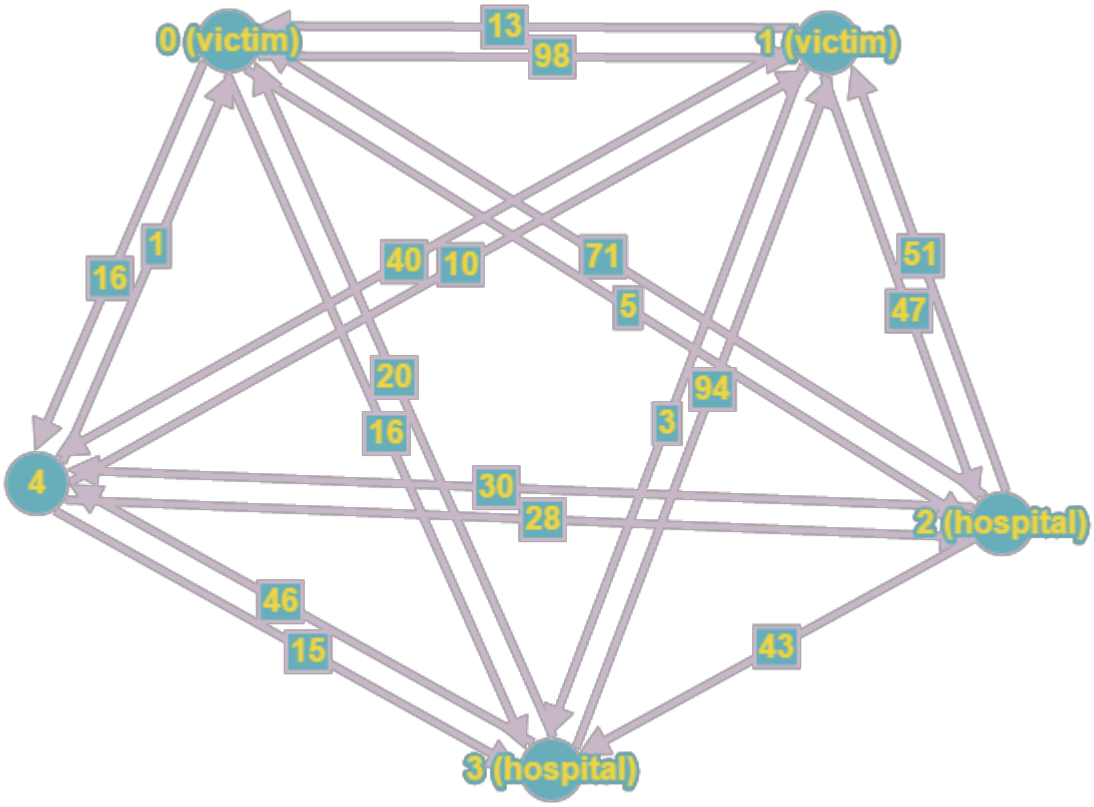
('*{x}**' mean zero or more of item x.)

There is at least one edge leaving from every node. The last line consists of a chronologically ordered series of emergency call events, where each event is represented by the number of the node at which the victim resides.

Example:

```
5
0 1 98 2 5 3 16 4 16
1 0 13 2 47 3 3 4 40
2 0 71 1 51 3 43 4 30
3 0 20 1 94 4 46
4 0 1 1 10 2 28 3 15
2
2 3
2
0 1
```

Representing the following weighted di-graph:



Output format:

Output is ordered by call event:

victim <*node V_0*>
<results for victim at node V_0>
…
victim <*node V_N*>
<results for victim at node V_N>

The results for a victim (at a given node) consist of details of the hospital(s) with the lowest cost round-trip path. If there is more than one shortest path, then these are output in ascending order of hospital node:

hospital <*hospital node H_X*>
<*shortest path from H_X to V_n to H_X*>

...
hospital <*hospital node H_Y*>
<*shortest path from H_Y to V_n to H_Y*>

A path is represented by the sequence of nodes traversed.

In the case that there is more than one minimum-cost round-trip between a victim at node V_i and a hospital at node H_j , the cost of these solutions is reported ***instead of*** these paths, i.e. output takes the following form:

...
victim <*node V_i >*

...
hospital <*hospital node H_j*>
multiple solutions cost <C_{min}>.

...

If there is no path from any hospital to a victim and back the output takes this form:

...
victim <*node V_N*>
cannot be helped

...

Example (assuming input data above):

```
victim 0
hospital 2
2 4 0 2
hospital 3
3 0 3
victim 1
hospital 3
3 0 4 1 3
```

Results for the victim at node 0 (the first call event) are followed with the results for the victim at node 1 (the second call event). *(It just happens in our example, that node number order matches call event order.)*

In the case of the first call event, there are two hospitals that give the same shortest round-trip cost, so both these paths are given, in ascending order of hospital-node number i.e. hospital 2 path first and then hospital 3 path.

In the case of the second call event, there is only one shortest path and this lowest round-trip cost is for the hospital at node 3.

Part two: extension

This part will NOT be marked by the Automarker.

Create a new version of your simulation called '`SimulatorTwo.java`' that incorporates some advanced features, such as the following:

- Ambulances are not always stationed at hospitals, and can take a victim to any hospital.
- Some/all ambulances can carry 2 or more patients.
- The number of ambulances is finite, as is the number of emergency room beds at each hospital.
- Ambulances and hospitals belong to particular groups (e.g. Medicross vs Mediclinic), and an ambulance may only operate between the hospitals of its group.
- Victim call events are interspersed with traffic report events that require changes to the graph weightings.
- A more explicit simulation of time such that emergency calls arrive at particular time and the time taken to fetch and deliver patients is tracked. You can assume that edge weightings represent a quantity of time. *(This feature is interesting if used in conjunction other possible features such as restrictions on the number of ambulances)*.

These examples are deliberately under specified. We want you to exercise your creativity.

Along with your code, you should submit a '`readme.txt`' document containing a brief description of the features that you have implemented.

Development requirements

For part I, you are required to make appropriate use of the following tools:

- javadoc, for documentation generation

For part two you are required make use of these additional tools:

- git, for source code management
- junit, for unit testing
- javadoc, for documentation generation
- make, for automation of compilation, documentation generation, unit testing and cleaning of files
- jacoco, for unit test code coverage

Submission requirements

Submit a .tar.gz compressed archive containing:

- Makefile
- src/
    - all source code
- bin/
    - all class files
- doc/
    - javadoc output

- test/
    - junit test classes
- coverage/
    - jacoco-generated coverage report
- readme.txt
    - description of the advanced features implemented for part two.

Do not submit the junit jar files, the jacoco jar files, or the git repository.

Marking guidelines

Artefact	***Aspect***	***Mark***
Solution to part one	Correctness	80
Solution to part two	Correctness	15
Development	Documentation – javadoc	5

END